Compiler Developments at Apple

Session 907

Compiler Developments at Apple

John Graziano
Engineering Manager,
Mac OS X Compiler Group

What We Will Cover

- GCC 3.1
    - New features
    - Converting projects
- Code Quality
- Compile Time

Gnu C Compiler

- Free software
- Supports ANSI C, C++ and Objective-C
- Robust C++ implementation
- Many, many years of work and testing
- “Reference” compiler for many developers
- Compiles all of Mac OS X

Latest Work in GCC 2.95

- Default compiler on 10.1.x
- Improvements to code generation
    - Code quality
    - Memory footprint
- Objective-C++
- Two-level Namespace

GCC 3.1

The Latest Mac OS X Compiler

New Features in GCC 3.1

- C99 Compliance
- C++ ANSI Compliance
- Integrated Preprocessor
- More and better optimizations
- C++ ABI Changes!

Apple Additions to GCC 3.1

- Improved PPC Code Generation
- New Precompiled Headers (C++!)
- Objective-C++
- Mach-O Support

Caution

- Wait for Jaguar GM to ship product on GCC 3.1
- Beta, beta, beta

Objective-C++

- Combines C++ and Objective-C
- Allows integration of C++ code with Cocoa applications
- New file extension: **.mm**
  - Legacy extension **.M** still supported (but discouraged)

Mixing C++ and ObjC Code

- ObjC declarations (e.g., **id foo, NSObject *bar**) can be intermixed with C++
- ObjC objects may point to C++ objects (and vice versa)
- ObjC message sends (e.g., **[myObj foo]**) can be intermixed with C++ expressions

ObjC++ Restrictions

- Object hierarchies cannot mix
- C++ classes cannot receive ObjC messages (or vice versa)
- Cannot statically allocate, **new** or **delete** ObjC objects

Objective-C++ Is For Real!

- Available in GCC 2.95 and 3.1
- Already in use on shipping applications

Code Generation

Measuring Code Quality

- Real-world code
- Benchmarks
- Test against other compilers
  - CodeWarrior
  - MrC
  - GCC 2.95

Real-World Code

- Large components of Mac OS X
- Measures larger factors than benchmarks
    - Memory usage
    - System interaction

Real-World Code: Examples

- QuickTime
- iTunes
- Mach Kernel
- Quartz and OpenGL
- Java VM

What Exactly Is a “Benchmark”?

- Collection of CPU-intensive routines
- Built for multiple platforms by multiple compilers
- Each test targets subset of compiler codegen
- Great yardstick for measuring basic optimization

Benchmarks We Use

- CPU 2000 (SPECMarks)
  - Large tests of system software
- ByteMarks
  - Obsolete, easily manipulated
- SkidMarks (Apple Internal)

SkidMarks Overview

- Developed by Apple's hardware group
- Real-world Macintosh code examples
- Smaller tests, focused on CPU usage
- 3 categories of tests
  - Integer
  - Floating Point
  - AltiVec™

SkidMarks Integer Tests

- MPEG
    - Open Source MPEG2 encoder
- PixBlend
    - Pixel blender used in Final Cut Pro and iDVD
- Ellipticrypt
    - Elliptical encryption routine
- Rijndael
    - NIST encryption algorithm

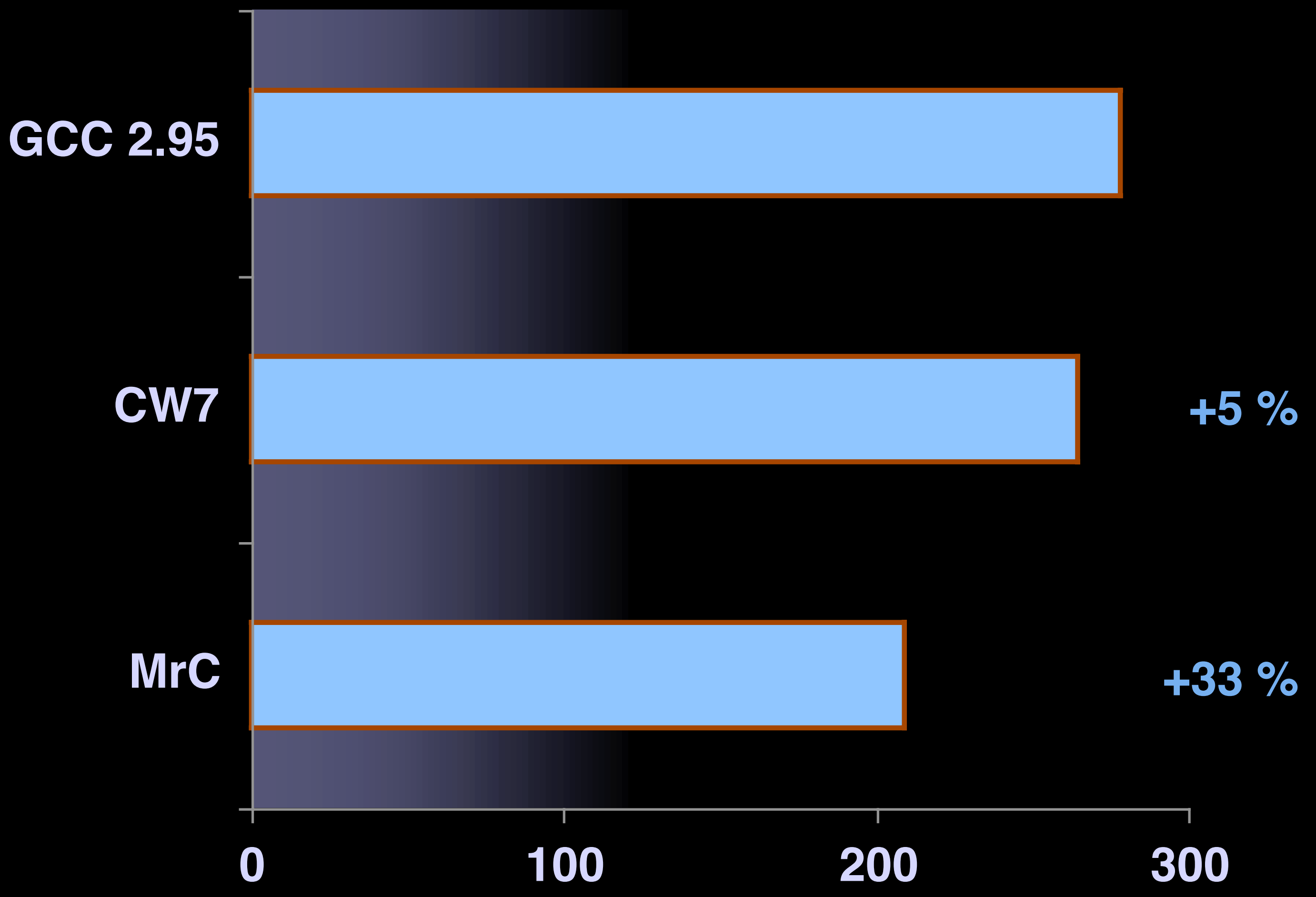
SkidMarks Floating Point Tests

- Q3
  - Quake3 math routine
- FFT
  - Fast Fourier transform
- VolInt
  - Volume integration of cubic region

SkidMarks AltiVec™ Tests

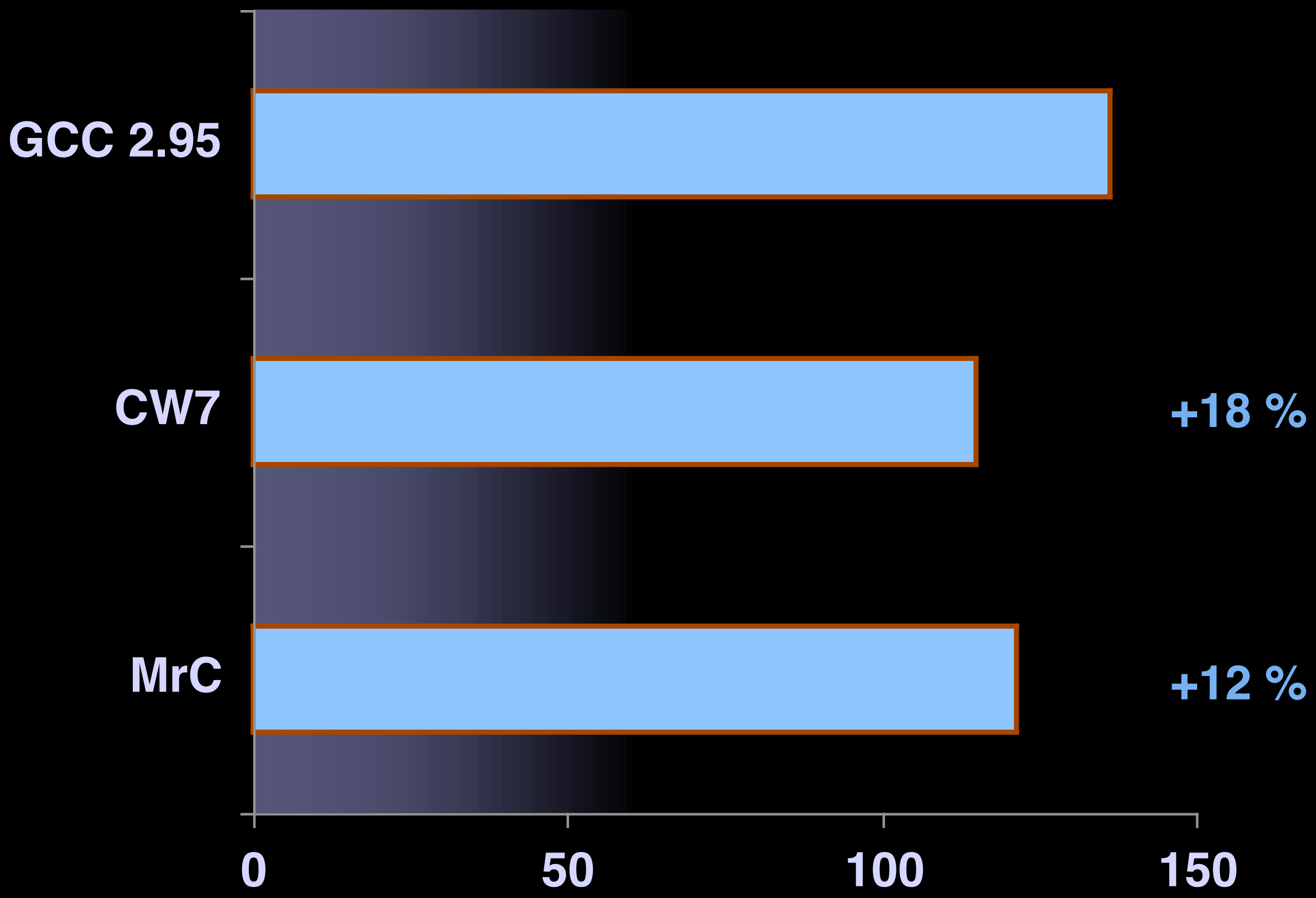
- Galaxy
  - Gravity calculation
- IDCT
  - Inverse Discreet Cosine Transform (QuickTime)
- BigMult
  - Multiplication of 4096-bit numbers

GCC 2.95: Integer*

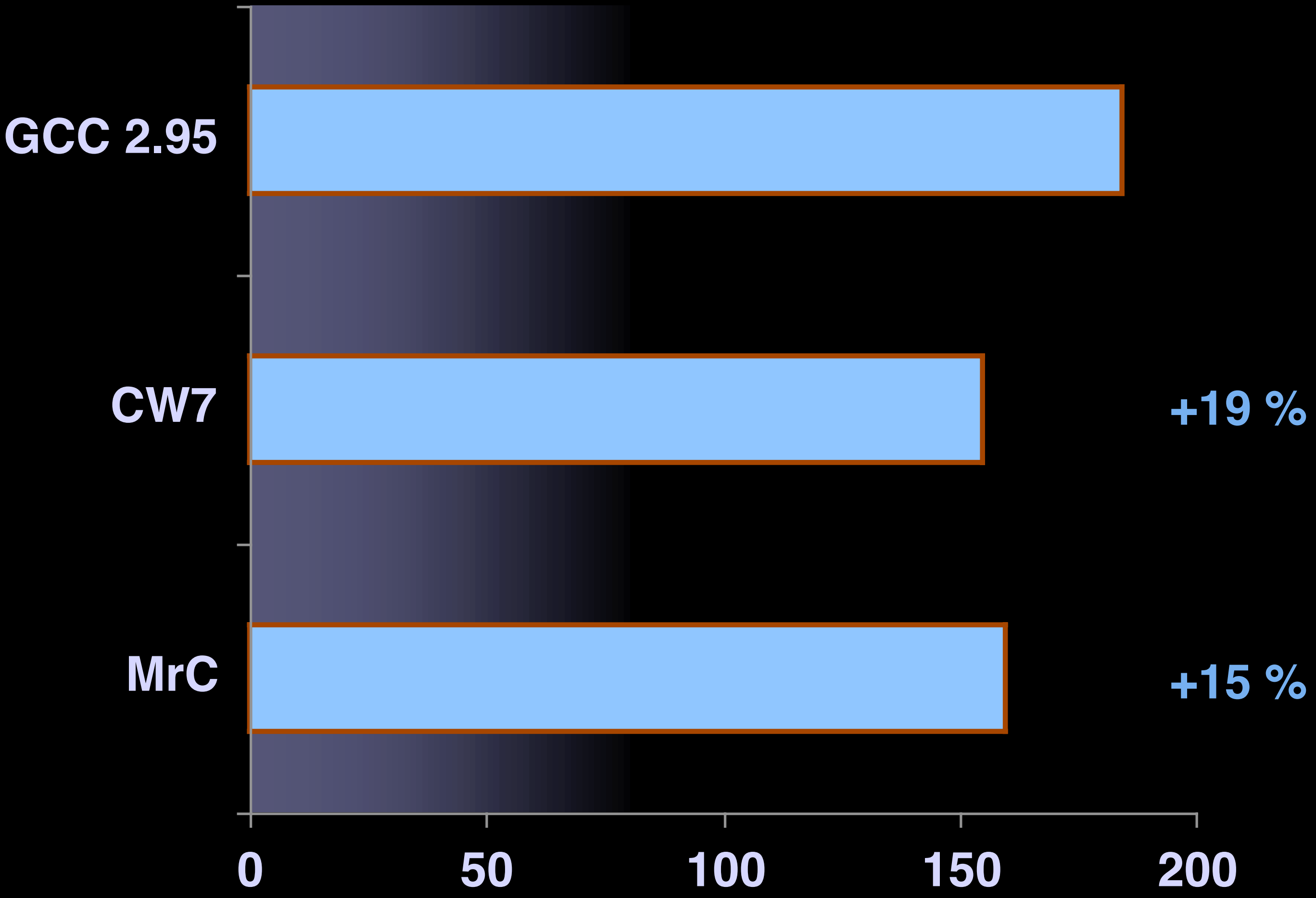


*Smaller is better

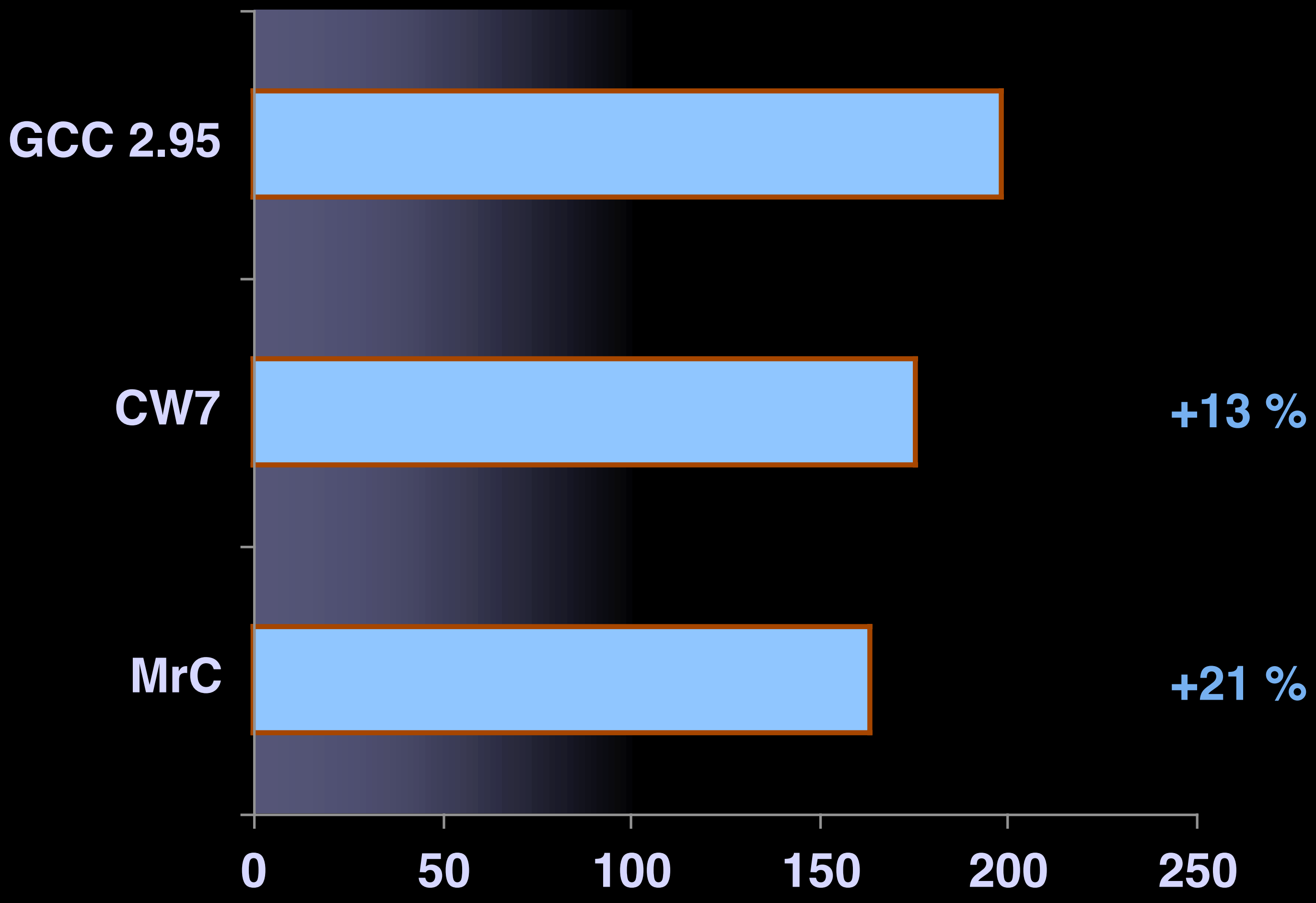
GCC 2.95: Floating Point



GCC 2.95: AltiVec™



GCC 2.95: Overall



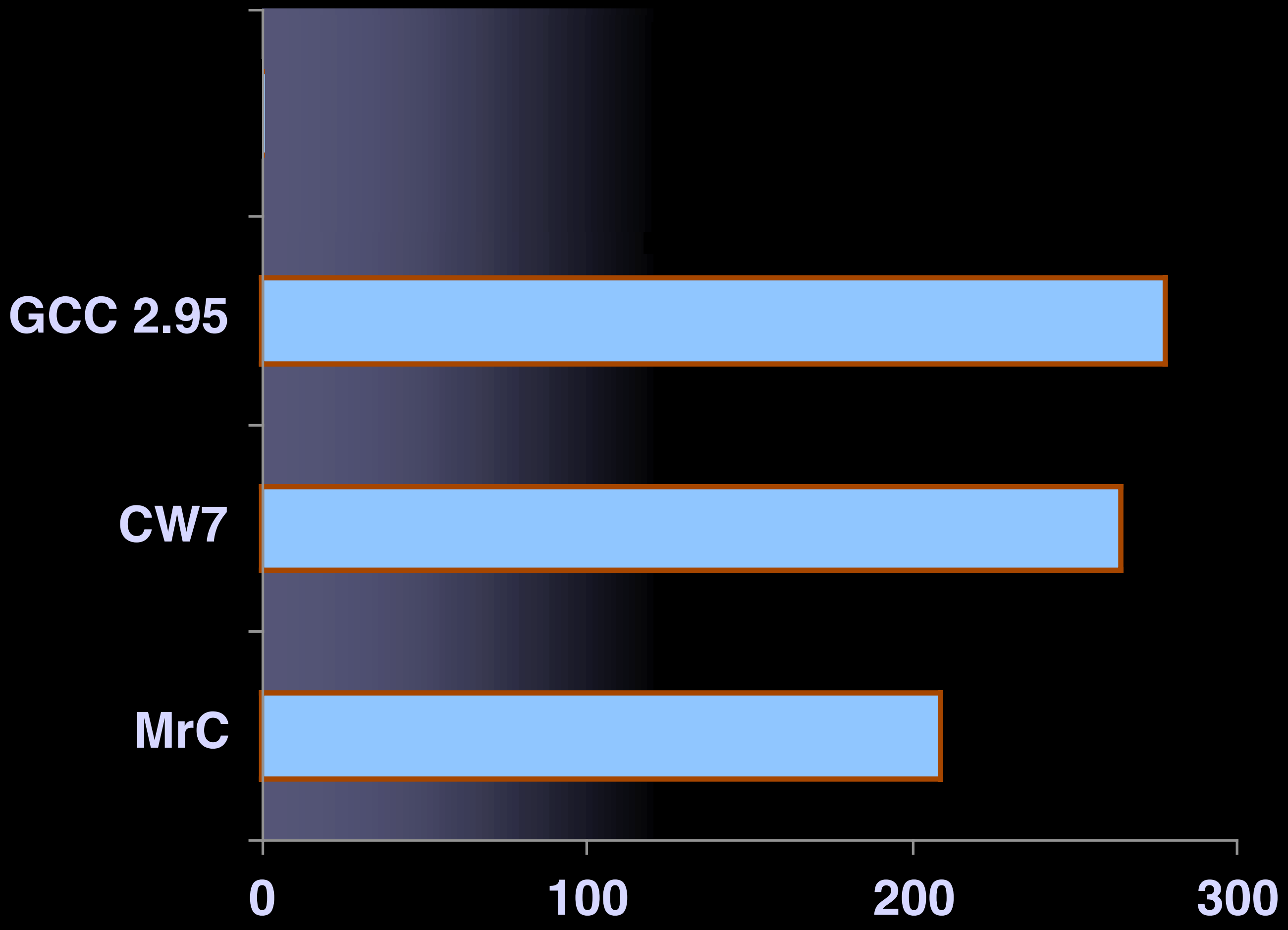
Codegen “Opportunities”

- NullStones
    - Identifies missing optimizations
- Head-to-head comparisons
    - Build identical code with several compilers
- Assembly inspection
- Compiler source code inspection

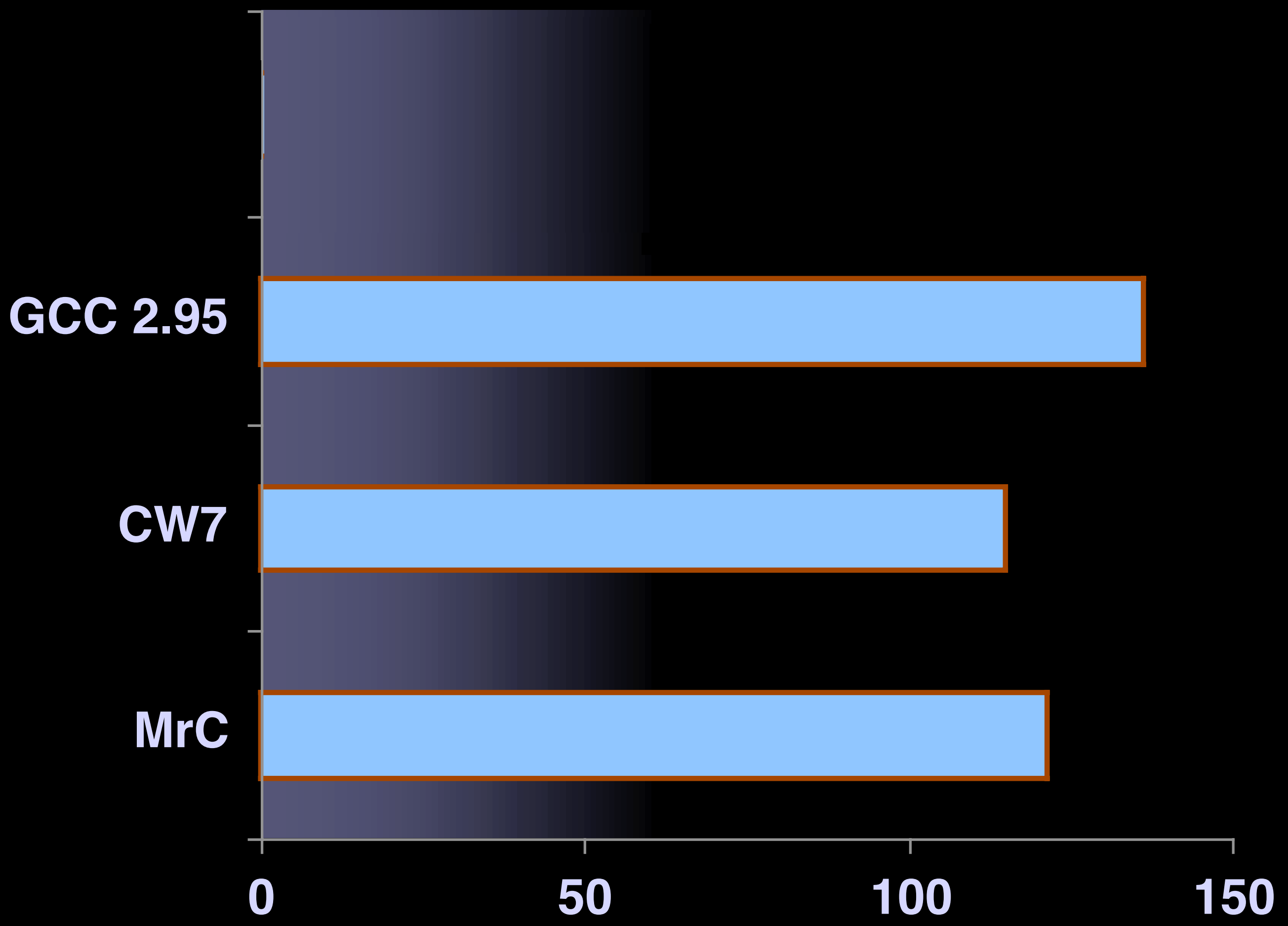
Codegen in GCC 3.1

- Forward inliner
- Dynamic, non-pic function calls
  - Removes library call indirection for executables
  - Saves 2 loads per call
- AltiVec™ and FP optimizations
- Continued incremental improvement

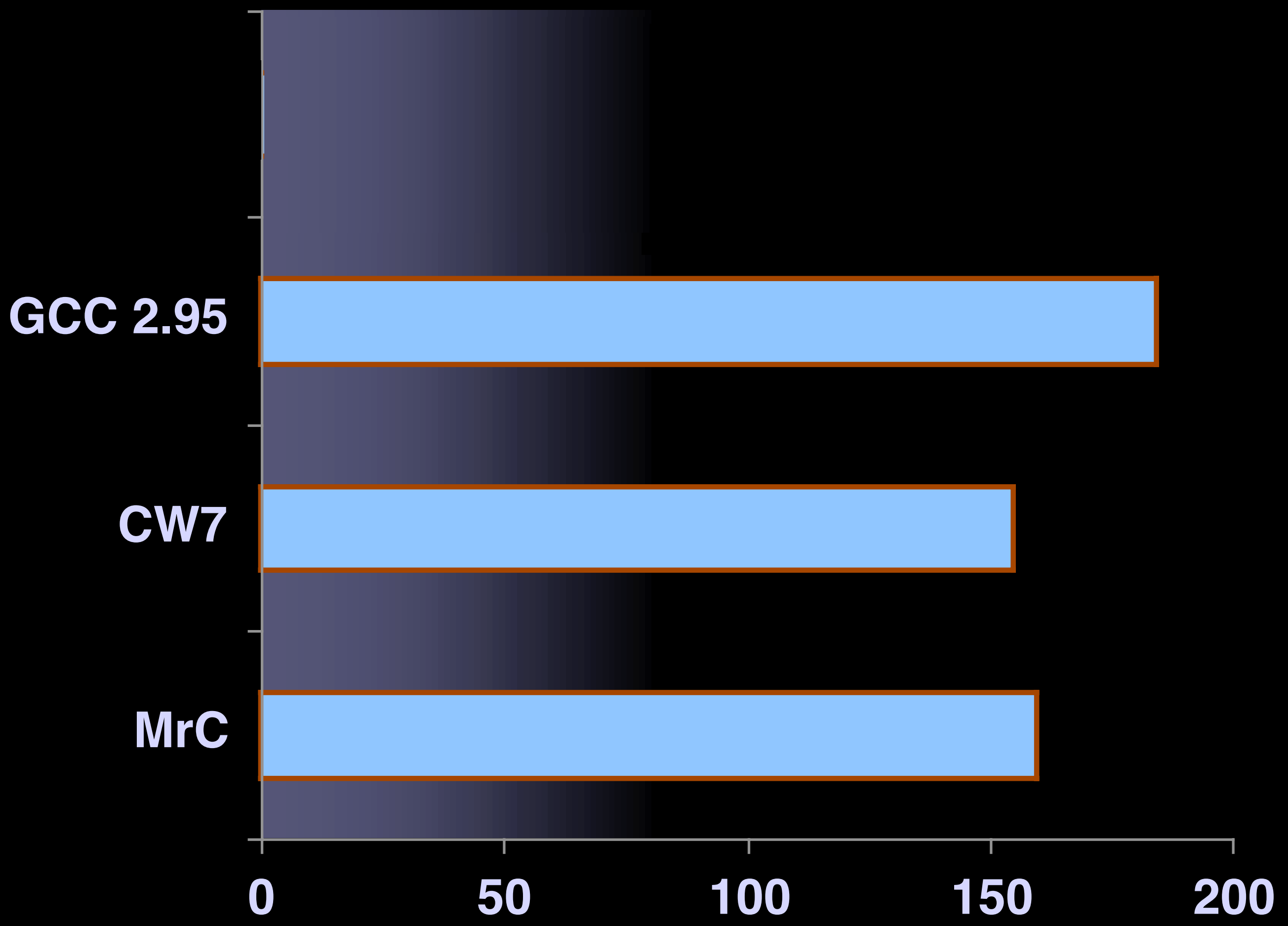
GCC 3.1: Integer



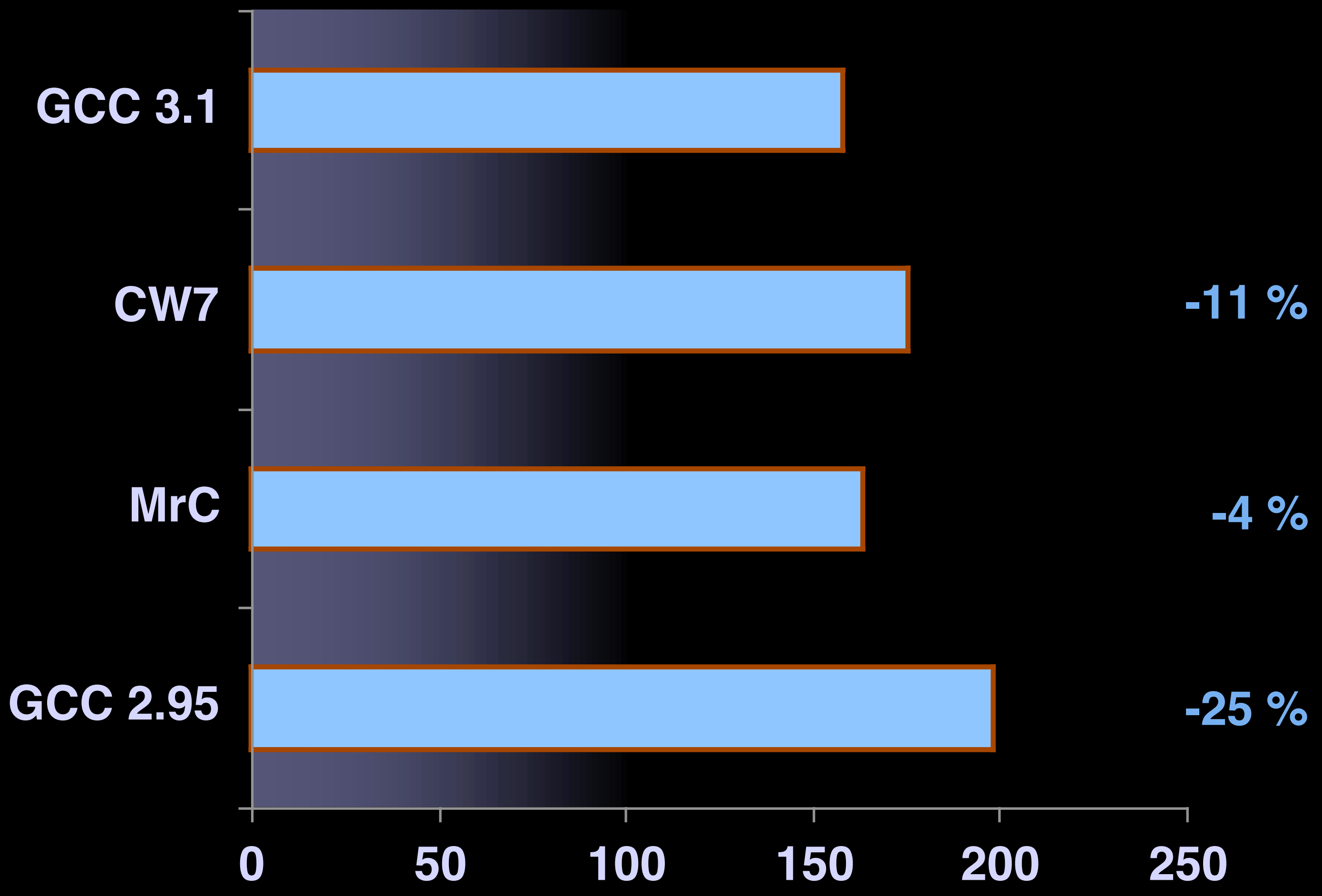
GCC 3.1: Floating Point



GCC 3.1: AltiVec™



GCC 3.1: Overall



Codegen and You . . .

- Optimize your code!!
    - Recommend **-Os** for all projects
- Measure, measure, measure . . .
    - Optimal settings depend on code

-Os: Optimize for Size

- Produces smallest binary size

- Performance roughly equivalent to **-O2**
  - No loop unrolling
  - No scheduling, register renaming
  - Limited inlining (only with inline keyword)

-mdynamic-no-pic

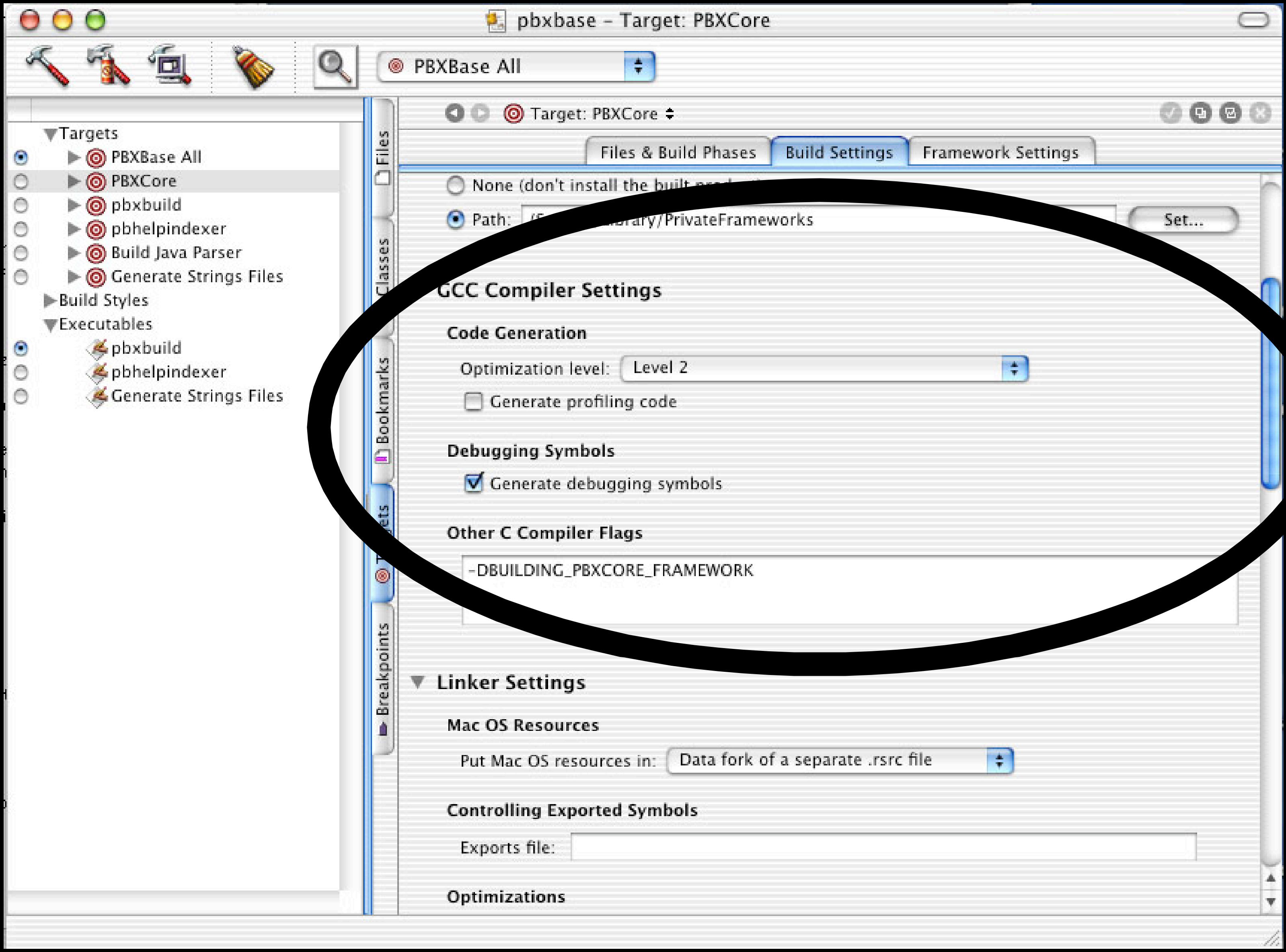
- New for GCC 3.1
- Generates indirect, non-position-independent function calls
  - Reduces code size by 10%
  - Increases code performance by 10%
- Default setting for PB Applications, Tools

Use only on executables

-O3 and Inlining

- **-O3** turns on automatic inlining
    - Including forward inlining
- **-O3** inlines using internal criteria
    - **inline** keyword only a hint
    - Use **-finline-limit** to set size of inlined functions

- Aggressive inlining can seriously bloat code



○ None (don't install the built prod...

● Path: /S... ...ibrary/PrivateFrameworks

GCC Compiler Settings

Code Generation

Optimization level: Level 2

□ Generate profiling code

Debugging Symbols

☑ Generate debugging symbols

Other C Compiler Flags

```
-DBUILDING_PBXCORE_FRAMEWORK
```

▼ Linker Settings

Mac OS Resources

Put Mac OS resources in: Data fork of a separate .rsrc file

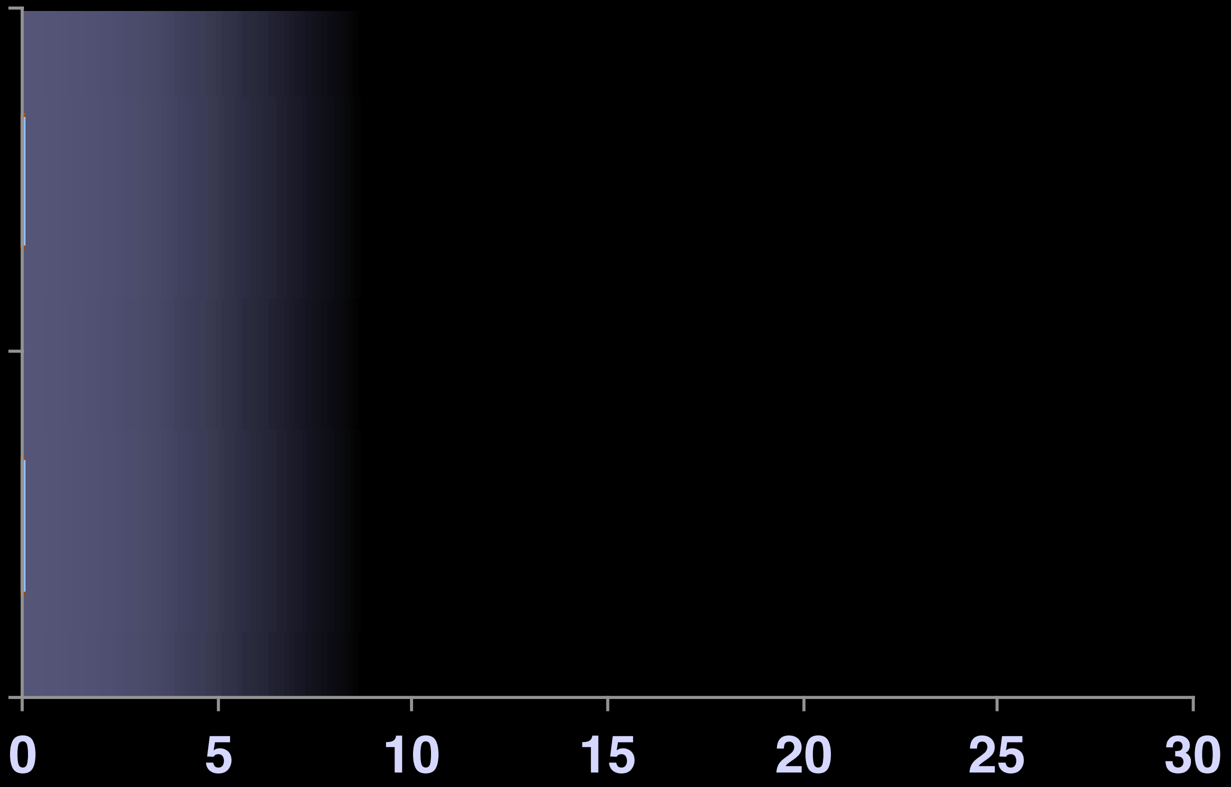
Controlling Exported Symbols

Exports file:

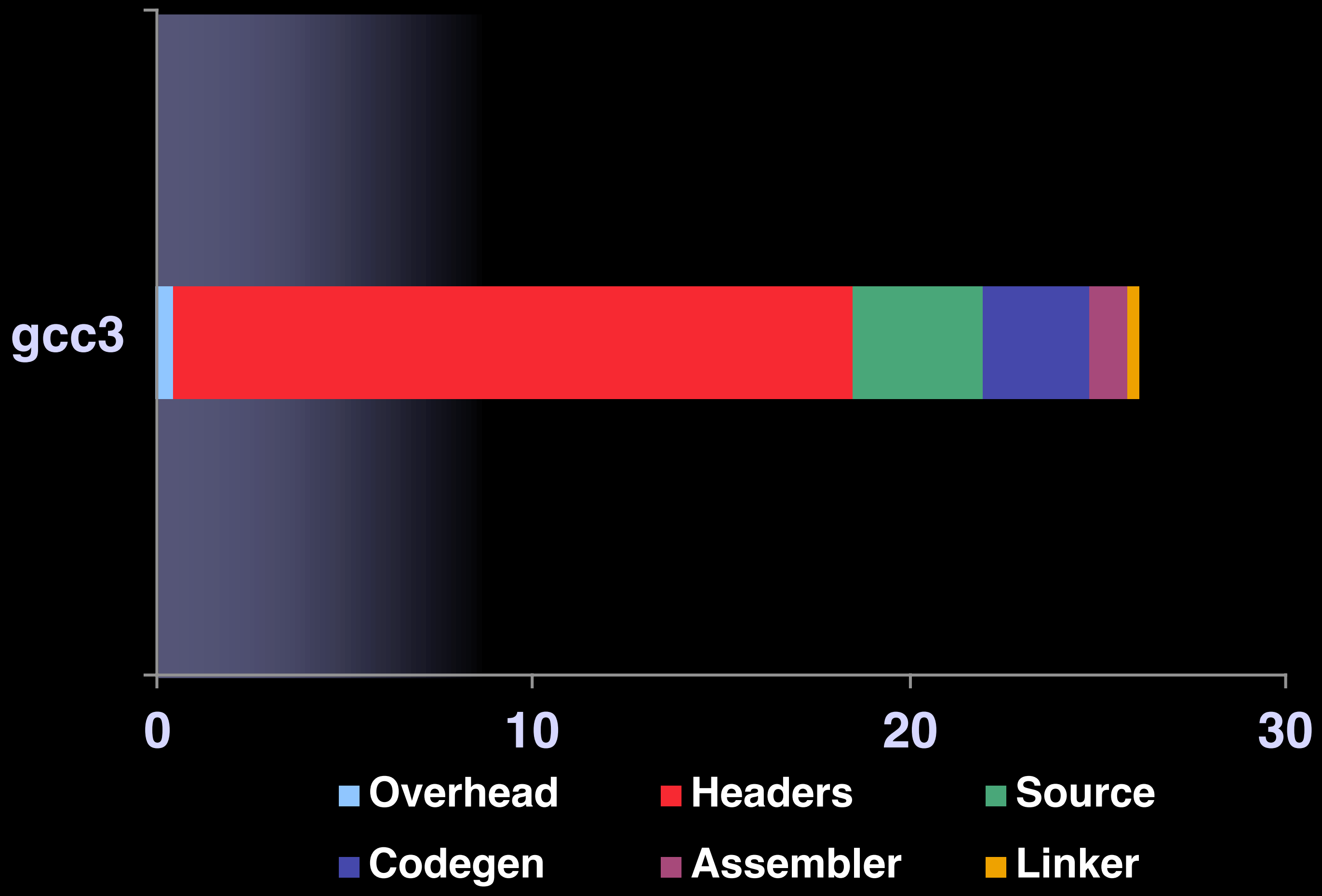
Optimizations

Build Time

C++ Build Time



Where Does the Time Go?



The Cost of Header Parsing

- I/0
  - Reading files
  - Searching
- Preprocessing
- Parsing Declarations
  - CPU usage
  - Memory allocation
- 100,000 lines of declarations in Carbon alone!

What about cpp-precomp?

- GCC 2.95 precomp mechanism
- Stores all headers in tokenized form
- Selectively unparses referenced declarations
- Good PB support

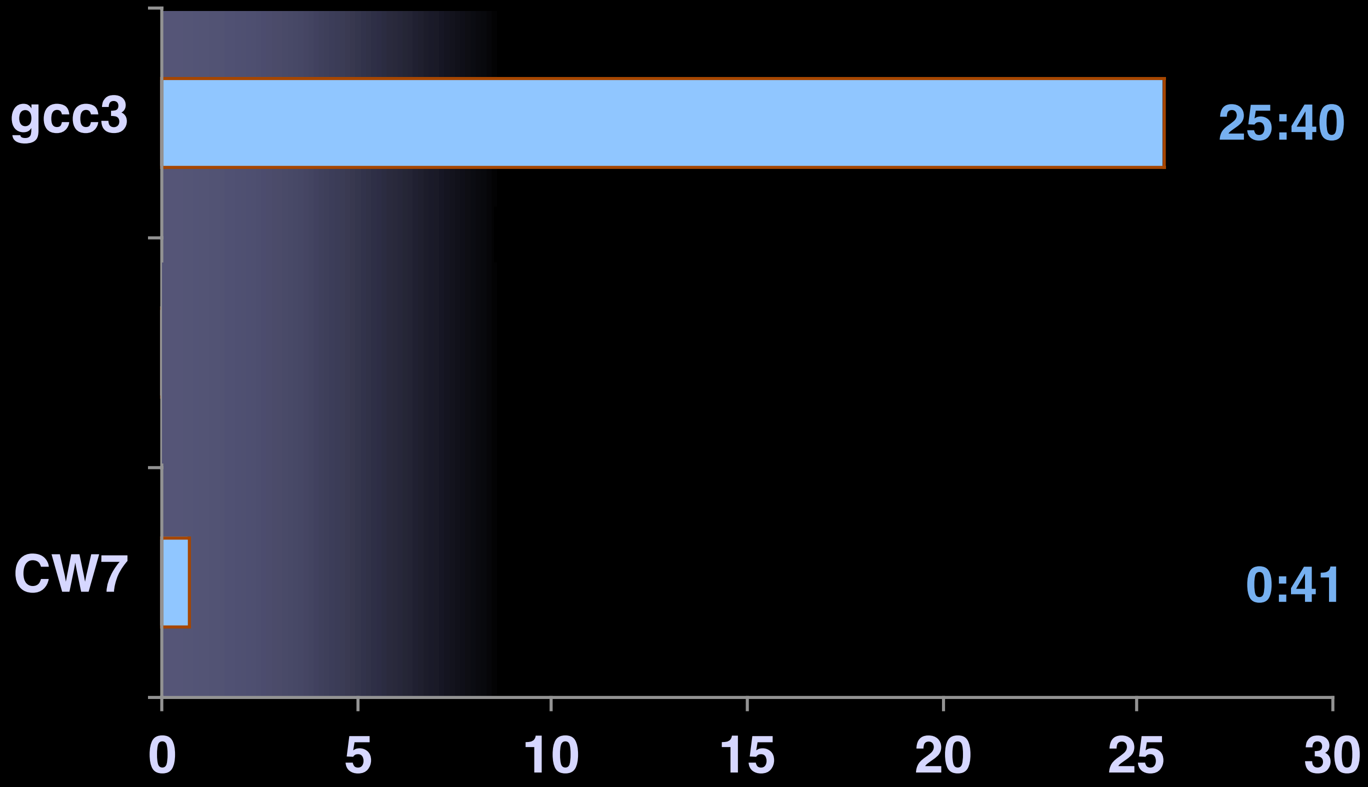
But . . .

- Cannot contain C++

Persistent Front End

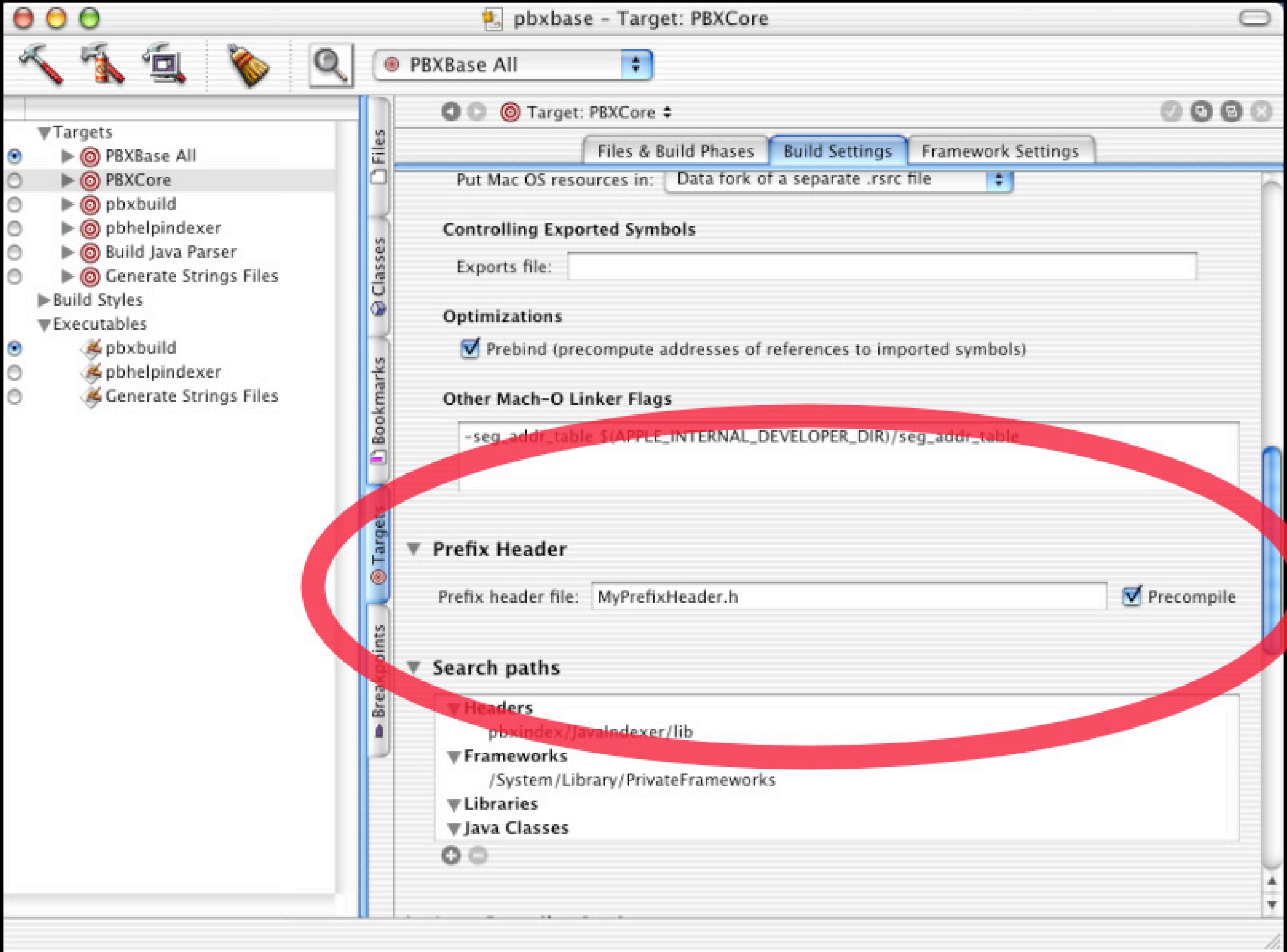
- Saves entire front end state to disk
    - File mapped in at known address
- Supports all C flavors
    - C, Objective-C, C++ Objective-C++

C++ Build Time–Progress



Compile Time Improvement

Up to 6x faster with GCC 3.1
and the
Persistent Front End



Put Mac OS resources in: Data fork of a separate .rsrc file

Controlling Exported Symbols

Exports file:

Optimizations

☑ Prebind (precompute addresses of references to imported symbols)

Other Mach-O Linker Flags

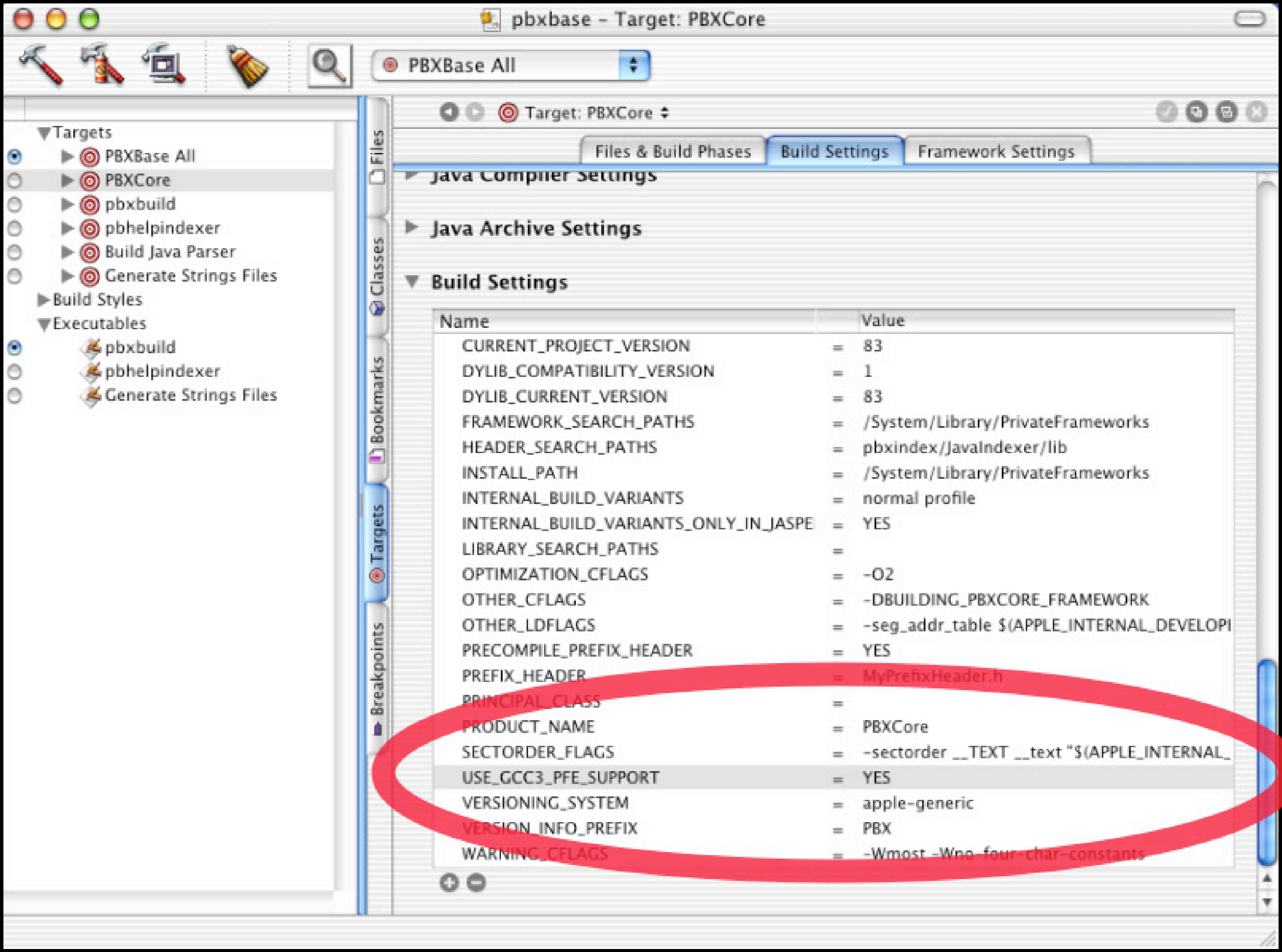
```
-seg_addr_table $(APPLE_INTERNAL_DEVELOPER_DIR)/seg_addr_table
```

Prefix Header

Prefix header file: MyPrefixHeader.h ☑ Precompile

Search paths

- Headers
  - pbxindex/JavaIndexer/lib
- Frameworks
  - /System/Library/PrivateFrameworks
- Libraries
- Java Classes



Build Settings

Name		Value
CURRENT_PROJECT_VERSION	=	83
DYLIB_COMPATIBILITY_VERSION	=	1
DYLIB_CURRENT_VERSION	=	83
FRAMEWORK_SEARCH_PATHS	=	/System/Library/PrivateFrameworks
HEADER_SEARCH_PATHS	=	pbxindex/JavaIndexer/lib
INSTALL_PATH	=	/System/Library/PrivateFrameworks
INTERNAL_BUILD_VARIANTS	=	normal profile
INTERNAL_BUILD_VARIANTS_ONLY_IN_JASPE	=	YES
LIBRARY_SEARCH_PATHS	=	
OPTIMIZATION_CFLAGS	=	-O2
OTHER_CFLAGS	=	-DBUILDING_PBXCORE_FRAMEWORK
OTHER_LDFLAGS	=	-seg_addr_table $(APPLE_INTERNAL_DEVELOPI
PRECOMPILE_PREFIX_HEADER	=	YES
PREFIX_HEADER	=	MyPrefixHeader.h
PRINCIPAL_CLASS	=	
PRODUCT_NAME	=	PBXCore
SECTORDER_FLAGS	=	-sectorder __TEXT __text "$(APPLE_INTERNAL_
USE_GCC3_PFE_SUPPORT	=	YES
VERSIONING_SYSTEM	=	apple-generic
VERSION_INFO_PREFIX	=	PBX
WARNING_CFLAGS	=	-Wmost -Wno-four-char-constants

PFE Today

- Header process speed increased 8-10x
- Full support of C, C++, ObjC, ObjC++
- Overall build speed increased as much as 6x

Moving to GCC 3.1

Changes

- New STL and libstdc++
    - iterators, exceptions
- Stricter ANSI compliance
- Better error checking
- New C++ ABI

New C++ ABI

Problem:

- Link fails with many undefined symbols

Cause:

- C++ library not recompiled with GCC 3.1

Fix:

- Rebuild all dependent modules with GCC 3.1

Namespaces

Problem:

- Compile fails with **symbol *sym* not in scope**

Cause:

- All C++ library classes are now in namespace **std**

Fix:

- Prefix symbol references with **std::** or add a **using std** directive

STL Changes

Problem:

Looser throw specifier…

Cause:

exception now defines an empty throw specifier for some methods

Fix:

Add empty throw specifiers to proper methods

Linking With libstdc++

Problem:

- Link fails with strange undefineds like **_gxx_personality**

Cause:

- GCC now requires an explicit link against **libstdc++**

Fix:

- Use **c++** command

Other Issues

- Type Agreement strictly enforced

  Use casts

- **cpp-precomp** doesn't like the new STL

  Use PFE

- **c++** recognizes operator names (**and**, **not_eq**, etc)

  Add **-fno-operator-names**

The Bottom Line

- C++ projects will require some code changes
- C and Objective-C should Just Work™

Still to Come in Jaguar

- Full sync with GCC 3.1 release
- Further codegen improvements
  - Speed
  - Size (esp. C++)
- PFE Tuning
- Great PFE support in Project Builder

GCC 3.1

- Better C++ Compliance
- Improved Code Quality
- Faster Compile Time

Use it now, ship with it for Jaguar

Technical Documentation

- /Developer/Documentation/DevTools
  - Compiler
  - Preprocessor
  - gdb
  - MachORuntime

Roadmap

908 Delivering with Project Builder: Hear about in-depth techniques	Hall 2 **Fri., 2:00pm**
FF015 Development Tools: Make your thoughts known	Room J1 **Fri., 3:30pm**
909 Debugging in Mac OS X: Learn about gdb and debugging techniques	Hall 2 **Fri., 5:00pm**

Who to Contact

Godfrey DiGiorgi
Technology Manager, Development Tools
ramarren@apple.com

Development Tools Engineering Feedback
macosx-tools-feedback@group.apple.com

Bug Reporting
http://developer.apple.com/bugreporter/

Q&A

Godfrey DiGiorgi
Technology Manager, Development Tools
ramarren@apple.com

http://developer.apple.com/wwdc2002/urls.html

WWDC2002

WWDC2002

WWDC2002